UIL Computer Science Written Test

2025 District

March 2025

General Directions (Please read carefully!)

1. DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO.
2. There are 40 questions on this contest exam. You will have 45 minutes to complete this contest.
3. All answers must be legibly written on the answer sheet provided. Indicate your answers in the appropriate blanks provided on the answer sheet. Clean erasures are necessary for accurate grading.
4. You may write on the test packet or any additional scratch paper provided by the contest director, but NOT on the answer sheet, which is reserved for answers only.
5. All questions have ONE and only ONE correct answer. There is a 2-point penalty for all incorrect answers.
6. Tests may not be turned in until 45 minutes have elapsed. If you finish the test before the end of the allotted time, remain at your seat and retain your test until told to do otherwise. You may use this time to check your answers.
7. If you are in the process of actually writing an answer when the signal to stop is given, you may finish writing that answer.
8. All provided code segments are intended to be syntactically correct, unless otherwise stated. You may also assume that any undefined variables are defined as used.
9. A reference to many commonly used Java classes is provided with the test, and you may use this reference sheet during the contest. AFTER THE CONTEST BEGINS, you may detach the reference sheet from the test booklet if you wish.
10. Assume that any necessary import statements for standard Java SE packages and classes (e.g., `java.util`, `System`, etc.) are included in any programs or code segments that refer to methods from these classes and packages.
11. NO CALCULATORS of any kind may be used during this contest.

Scoring

1. Correct answers will receive **6 points**.
2. Incorrect answers will lose **2 points**.
3. Unanswered questions will neither receive nor lose any points.
4. In the event of a tie, the student with the highest percentage of attempted questions correct shall win the tie.

STANDARD CLASSES AND INTERFACES – SUPPLEMENTAL REFERENCE

package java.lang

```
class Object
  boolean equals(Object anotherObject)
  String toString()
  int hashCode()

interface Comparable<T>
  int compareTo(T anotherObject)
```
Returns a value < 0 if `this` is less than `anotherObject`.
Returns a value = 0 if `this` is equal to `anotherObject`.
Returns a value > 0 if `this` is greater than `anotherObject`.

```
class Integer implements Comparable<Integer>
  Integer(int value)
  int intValue()
  boolean equals(Object anotherObject)
  String toString()
  String toString(int i, int radix)
  int compareTo(Integer anotherInteger)
  static int parseInt(String s)

class Double implements Comparable<Double>
  Double(double value)
  double doubleValue()
  boolean equals(Object anotherObject)
  String toString()
  int compareTo(Double anotherDouble)
  static double parseDouble(String s)

class String implements Comparable<String>
  int compareTo(String anotherString)
  boolean equals(Object anotherObject)
  int length()
  String substring(int begin)
```
Returns `substring(begin, length())`.
```
  String substring(int begin, int end)
```
Returns the substring from index `begin` through index `(end - 1)`.
```
  int indexOf(String str)
```
Returns the index within `this` string of the first occurrence of `str`. Returns `-1` if `str` is not found.
```
  int indexOf(String str, int fromIndex)
```
Returns the index within `this` string of the first occurrence of `str`, starting the search at `fromIndex`. Returns `-1` if `str` is not found.
```
  int indexOf(int ch)
  int indexOf(int ch, int fromIndex)
  char charAt(int index)
  String toLowerCase()
  String toUpperCase()
  String[] split(String regex)
  boolean matches(String regex)
  String replaceAll(String regex, String str)

class Character
  static boolean isDigit(char ch)
  static boolean isLetter(char ch)
  static boolean isLetterOrDigit(char ch)
  static boolean isLowerCase(char ch)
  static boolean isUpperCase(char ch)
  static char toUpperCase(char ch)
  static char toLowerCase(char ch)

class Math
  static int abs(int a)
  static double abs(double a)
  static double pow(double base, double exponent)
  static double sqrt(double a)
  static double ceil(double a)
  static double floor(double a)
  static double min(double a, double b)
  static double max(double a, double b)
  static int min(int a, int b)
  static int max(int a, int b)
  static long round(double a)
  static double random()
```
Returns a `double` greater than or equal to `0.0` and less than `1.0`.

package java.util

```
interface List<E>
class ArrayList<E> implements List<E>
  boolean add(E item)
  int size()
  Iterator<E> iterator()
  ListIterator<E> listIterator()
  E get(int index)
  E set(int index, E item)
  void add(int index, E item)
  E remove(int index)

class LinkedList<E> implements List<E>, Queue<E>
  void addFirst(E item)
  void addLast(E item)
  E getFirst()
  E getLast()
  E removeFirst()
  E removeLast()

class Stack<E>
  boolean isEmpty()
  E peek()
  E pop()
  E push(E item)

interface Queue<E>
class PriorityQueue<E>
  boolean add(E item)
  boolean isEmpty()
  E peek()
  E remove()

interface Set<E>
class HashSet<E> implements Set<E>
class TreeSet<E> implements Set<E>
  boolean add(E item)
  boolean contains(Object item)
  boolean remove(Object item)
  int size()
  Iterator<E> iterator()
  boolean addAll(Collection<? extends E> c)
  boolean removeAll(Collection<?> c)
  boolean retainAll(Collection<?> c)

interface Map<K,V>
class HashMap<K,V> implements Map<K,V>
class TreeMap<K,V> implements Map<K,V>
  Object put(K key, V value)
  V get(Object key)
  boolean containsKey(Object key)
  int size()
  Set<K> keySet()
  Set<Map.Entry<K, V>> entrySet()

interface Iterator<E>
  boolean hasNext()
  E next()
  void remove()

interface ListIterator<E> extends Iterator<E>
  void add(E item)
  void set(E item)

class Scanner
  Scanner(InputStream source)
  Scanner(String str)
  boolean hasNext()
  boolean hasNextInt()
  boolean hasNextDouble()
  String next()
  int nextInt()
  double nextDouble()
  String nextLine()
  Scanner useDelimiter(String regex)
```

STANDARD CLASSES AND INTERFACES – SUPPLEMENTAL REFERENCE

Package java.util.function

```
Interface BiConsumer<T,U>
  void accept(T t, U u)

Interface BiFunction<T,U,R>
  R apply(T t, U u)

Interface BiPredicate<T,U>
  boolean test(T t, U u)

Interface Consumer<T>
  void accept(T t)

Interface Function<T,R>
  R apply(T t)

Interface Predicate<T>
  boolean test(T t)

Interface Supplier<T>
  T get()
```

UIL COMPUTER SCIENCE WRITTEN TEST – 2025 DISTRICT

Note: Correct responses are based on **Java SE Development Kit 22 (JDK 22)** from Oracle, Inc. All provided code segments are intended to be syntactically correct, unless otherwise stated (e.g., "error" is an answer choice) and any necessary Java SE 22 Standard Packages have been imported. Ignore any typographical errors and assume any undefined variables are defined as used. **For all output statements, assume that the System class has been statically imported using:** **`import static java.lang.System.*;`**

Question 1

Which of the following is not equivalent to the expression 2523_7 + $AB7_{16}$?

A) 2856_{11} **B)** 7153_8 **C)** 1161_{15} **D)** 104231_5 **E)** All are equivalent

Question 2

What is output by the code to the right?

A) 1 **B)** 9 **C)** 7 **D)** 0

E) There is no output due to a compile error.

```
out.println(1 + 2 * 3 - 4 / 5);
```

Question 3

What is output by the code to the right?

A) abcde **B)** abc

C) ABCDE **D)** ABC

E) There is no output due to a runtime error.

```
out.printf("%3S","abcde");
```

Question 4

What is output by the code to the right?

A) untDoot **B)** ntDookt

C) ntDookn **D)** untDoon

E) There is no output due to a runtime error.

```
String str = "CountDooku";
str = str.substring(0,8);
str += str.charAt(3);
out.println(str.substring(2));
```

Question 5

What is output by the code to the right?

A) true **B)** false

C) There is no output due to a syntax error.

```
boolean a = false;
boolean b = true;
a |= !b & a ^ b & !a;
out.println(a);
```

Question 6

What is output by the code to the right?

A) 9 **B)** 8 **C)** 8.0 **D)** 9.0

E) There is no output due to a runtime error.

```
int y = 9;
double x = 8.0;
out.print(Math.max(y,x));
```

Question 7

What is output by the code to the right?

A) 22 32 **B)** 20 30

C) 21 31 **D)** 19 29

E) There is no output due to a runtime error.

```
int i = 0, ii = 10, n = 0;
for(;i <= ii;) {
    ii++;
    i += ii / 10;
    n++;
}
out.println(n+" "+ii);
```

Question 8

What is output by the code to the right?

A) 101 **B)** 8 **C)** 36 **D)** 6

E) There is no output due to a runtime error.

```
int a = 34 + 21 & 9;
int b = a | 39 % 7;
a ^= b * 9 / 5;
out.println(a);
```

Question 9

What is output by the code to the right?

A) `9` **B)** `611` **C)** `610` **D)** `10`

E) There is no output due to a runtime error.

```
int i = 9;
if(i < 10)
    i++;
else if(i < 10)
    i--;
else
    out.print(6);
out.println(i);
```

Question 10

What is the output by the code to the right?

A) `1` **B)** `3` **C)** `5` **D)** `2`

E) There is no output due to a runtime error.

```
int[] i = new int[] {
    3, 2, 5, 4, 1, 0
};
int j = 3;
for(int k = 0; k < 25; k++)
    j = i[j];
out.println(i[j]);
```

Question 11

Which of the following packages contains the `Scanner` class?

A) `java.lang.*` **B)** `java.awt.*` **C)** `java.util.*` **D)** `java.io.*` **E)** None of the above.

Question 12

What is output by the code to the right?

A) `109` **B)** `97`

C) `128` **D)** `115`

E) There is no output due to a runtime error.

```
int sum = 1;
for(int y = 0; y < 12; y++) {
    sum += y;
    for(int x = 0; x < y / 2; x++)
        sum ++;
}
out.println(sum);
```

Question 13

What is the order of precedence for the operators to the right?

A) `III, IV, II, I` **B)** `IV, III, I, II`

C) `IV, III, II, I` **D)** `III, II, IV, I`

E) `III, II, I, IV`

```
I. ?:
II. + (additive)
III. %
IV. >>>
```

Question 14

What is output by the code to the right?

A) `8` **B)** `64` **C)** `4` **D)** `32`

E) There is no output due to a compile error.

```
out.println(Double.BYTES);
```

Question 15

What is the output by the code to the right?

A) `[-17, 451, 1]`

B) `[212, 451, -17]`

C) `[1, 451, -17]`

D) `[-17, 451, 212]`

E) There is no output due to a compile error.

```
ArrayList<Integer> a;
a = new ArrayList<Integer>();
a.add(1);
a.add(212);
a.add(451);
a.remove(1);
a.add(-17);
out.println(a);
```

Question 16

What is the output by the code to the right?

A) `Greater` **B)** `Not Greater`

C) `-1` **D)** `1`

E) There is no output due to a compile error.

```
out.print("instanceof".compareTo("int")
   > 4 ? "Greater" : "Not Greater");
```

Question 17

Which of the following values is a possible value that the expression to the right may resolve to?

A) `22.8` **B)** `23` **C)** `36.0`

D) A and B

E) A and C

Question 18

Using interval notation where a '[' or ']' represents an inclusive value, and a '(' or ')' represents an exclusive value, which of the following denotes the set of all possible values that can be returned by the method call to the right?

A) $[21, 57)$ **B)** $[21, 57]$

C) $(21, 57]$ **D)** $[21, 36)$

E) None of the above.

```
(Math.random() * 36) + 21
```

Question 19

Which of the following best classifies the graph to the right?

A) Undirected, Weighted, and Acyclic.

B) Undirected and Unweighted.

C) Directed and Unweighted.

D) Directed, Weighted, and Acyclic.

E) Directed, Weighted, and Cyclic.

Question 20

Which of the following algorithms are guaranteed to produce the shortest path between two specific nodes in the graph to the right?

A) Dijkstra's Algorithm

B) Bellman-Ford Algorithm

C) Floyd-Warshall AlgorithmW

D) Both B and C

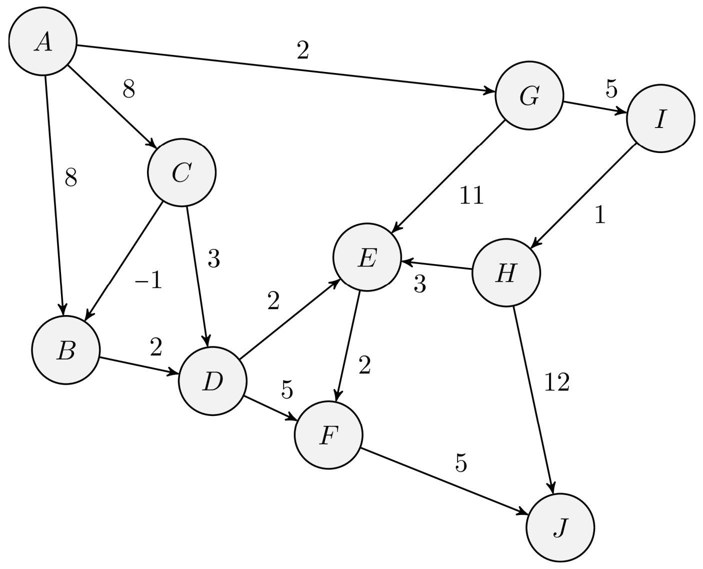
E) All of the above.

Question 21

What is the shortest path between the node labeled A and the node labeled J in the graph to the right?

A) 18 **B)** 19 **C)** 20 **D)** 21

E) None of the above.



Question 22

How many ordered pairs of A, B, and C, make the boolean expression to the right resolve to true?

A) 0 **B)** 3 **C)** 5 **D)** 7 **E)** 8

$$\overline{A \oplus \overline{C * B}} + \overline{A \oplus \bar{C} \oplus B}$$

Question 23

What is output by the code to the right?

A) `2042 0` **B)** `1438 2042`

C) `0 1438` **D)** `2042 1438`

E) There is no output due to a runtime error.

```
int x = 1438;
int y = 2042;
x ^= y ^= x ^= y;
out.println(x+" "+y);
```

Question 24

What could replace **<1*>** in the code to the right so that the `A` class compiles and functions as intended?

A) `this.y = n * 2;`
`y = n;`

B) `super(n * 2, n);`

C) `super(n * 2) ;`
`y = n;`

D) `super(n * 2);`
`super(n);`

E) More than one of the above.

Question 25

What is the output by the line marked `//q25` in the client code to the right?

A) `3 8` **B)** `3 4`

C) `6 8` **D)** `6 4`

E) There is no output due to a compile error.

Question 26

Assuming any errors above the line marked `//q26` have been corrected, what is the output by the line marked `//q26` in the client code to the right?

A) `8`

B) `10`

C) Output cannot be determined until runtime.

D) There is no output due to a compile error.

E) There is no output due to a runtime error.

Question 27

Assuming any errors above the line marked `//q27` have been corrected, what is the output by the line marked `//q27` in the client code to the right?

A) `12`

B) `13`

C) Output cannot be determined until runtime.

D) There is no output due to a compile error.

E) There is no output due to a runtime error.

```
class A{
    int y;

    public A(int n) {
        y = n;
    }

    public int get() {
        return y;
    }
}
class B extends A{
    int y;

    public B(int n) {
        <1*>
    }

    public void add() {
        y++;
    }
}
///////////client code//////////
A a = new A(3);
B b = new B(4);
String s = "";
s += a.get()+" ";
out.println(s+b.get()); //q25
b.add();
b.add();
out.println(b); //q26
A c = new B(6);
c.add();
out.println(c.get()); //q27
```

Question 28

Which of the following are equivalent to the Logic Circuit to the right?

A) $\overline{\bar{A} + B * (C \oplus \bar{D})}$ **B)** $\overline{\overline{\bar{A} + \bar{B} * (C \oplus \bar{D})}}$

C) $\overline{(A * \bar{B}) * (C \oplus \bar{D})}$ **D)** $\overline{(A * B) * (C \oplus \bar{D})}$

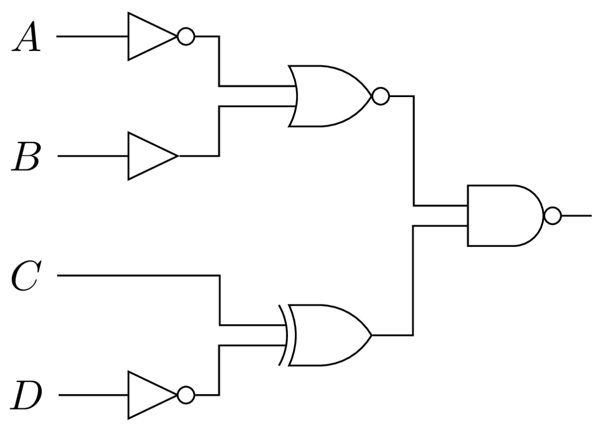
E) $\overline{A * \bar{B}} + \overline{C \oplus \bar{D}}$ **F)** $\overline{A * B} + \overline{C \oplus \bar{D}}$

G) $\bar{A} + B + C * \bar{D} + \bar{C} * D$ **H)** $\bar{A} + \bar{B} + C * \bar{D} + \bar{C} * D$

J) Options A, C, E, and G.

K) Options B, D, F, and H.

L) None of the above.



Question 29

The process of going from Option A to Option C in the previous question is an example of which Boolean Algebra Identity? A copy of the options has been provided for you below:

Question 28, Option A: $\overline{\overline{\bar{A} + B} * (C \oplus \bar{D})}$

Question 28, Option C: $\overline{(A * \bar{B}) * (C \oplus \bar{D})}$

A) Law of Absorption **B)** Exclusive NOR Law **C)** DeMorgan's Law **D)** Double Negative Law **E)** Disappearing Opposite

Question 30

Consider the following snippets of Option F and Option H from Question 28. The process of going from Snippet 1 to Snippet 2 is an example of which Boolean Algebra Identity?

Snippet 1 (Question 28, Option F): $\overline{C \oplus \bar{D}}$

Snippet 2 Question 28, Option H): $C * \bar{D} + \bar{C} * D$

A) Law of Absorption **B)** Exclusive NOR Law **C)** DeMorgan's Law **D)** Double Negative Law **E)** Disappearing Opposite

Question 31

What is output by the code to the right?

A)
```
[3, 6, 9]
[2, 5, 8]
[1, 4, 7]
```

B)
```
[3, 4, 1]
[2, 5, 2]
[1, 6, 3]
```

C)
```
3 6 9
2 5 8
1 4 7
```

D)
```
7 4 1
8 5 2
9 6 3
```

E)
```
9 2 7
4 5 8
3 6 1
```

F) There is no output due to a runtime error.

```
int[][] m = new int[][] {
     {1, 2, 3},
     {4, 5, 6},
     {7, 8, 9}
};

int n = m.length;
for (int i = 0; i < n / 2; i++) {
     for (int j = i; j < n - i - 1; j++) {
          int temp = m[i][j];
          m[i][j] = m[j][n - 1 - i];
          m[j][n - 1 - i] =
               m[n - 1 - i][n - 1 - j];
          m[n - 1 - i][n - 1 - j] =
               m[n - 1 - j][i];
          m[n - 1 - j][i] = temp;
     }
}

for(int i = 0; i < n; i++) {
     out.println(Arrays.toString(m[i])
          .replaceAll("[\\[\\],]", ""));
}
```

Question 32

What is the output by the line marked `//q32` in the client code to the right?

A) `0` **B)** `1` **C)** `2` **D)** `4` **E)** `5`

Question 33

What is the output by the line marked `//q33` in the client code to the right?

A) `15` **B)** `20` **C)** `21` **D)** `35` **E)** `56`

Question 34

Which of the following calls to the function rec would be equivalent to the output by the line marked `//q34` in the client code to the right?

A) `rec(6,11)` **B)** `rec(11,6)`

C) `rec(7,12)` **D)** `rec(12,7)`

E) None of the above.

```
public static int rec(int r, int c) {
     if(c > r) {
          return 0;
     } else if(c <= 1 || r <= 1) {
          return 1;
     }
     return rec(r-1, c) + rec(r-1, c-1);
}

///////////// client code /////////////
out.println(rec(3,2)); //q32
out.println(rec(8,5)); //q33
out.println(rec(10,6) + rec(10,5)); //q34
```

Question 35

The function `rec(r,c)` from the previous 3 questions is an implementation of which well-known recursive formula?

A) Fibonacci Sequence **B)** Pascal's Triangle **C)** Fast Exponentiation **D)** Factorial of a Number **E)** Triangular Numbers

Question 36

Which of the following could replace **<1*>** in the code to the right and have the code compile without error?

A) `public` **B)** `protected` **C)** `private`

D) Replace all instances of **<1*>** with nothing (delete it and leave the space blank).

E) More than one of the above.

Question 37

Which of the following could replace **<2*>** in the code to the right so that the code execution only takes the respective branch if `lock` is an example of the `BakeryLock` or `FilterLock` classes, respectively?

A) `extends` **B)** `implements`

C) `instanceof` **D)** `super`

E) More than one of the above.

Question 38

Which of the following could replace **<3*>** in the code to the right so that when the client code is executed, the following is printed:

```
BakeryLock lock(10)
func1()
BakeryLock unlock(10)
FilterLock cs.lock()
func2()
FilterLock cs.unlock()
```

A) `((FilterLock) lock).func2();`

B) `FilterLock.class.cast(lock).func2();`

C)
```
try {
     lock.getClass().getMethod("func2").
          invoke(lock);
} catch (Exception e) {
     e.printStackTrace();
}
```

D) Options A and C.

E) All of the above.

F) None of the above.

```
interface Lock {
    public void lock();
    public void unlock();
}

class BakeryLock implements Lock {
    private int n;

    public BakeryLock(int n) {
        this.n = n;
    }

    <1*> void lock() {
        out.printf(
            "BakeryLock lock(%d)\n", n);
    }

    <1*> void unlock() {
        out.printf(
            "BakeryLock unlock(%d)\n", n);
    }

    public void func1() {
        out.println("func1()");
    }
}

class FilterLock implements Lock {
    private String name;

    public FilterLock(String s) {
        this.name = s;
    }

    <1*> void lock() {
        out.printf(
            "FilterLock %s.lock()\n", name);
    }

    <1*> void unlock() {
        out.printf(
            "FilterLock %s.unlock()\n", name);
    }

    public void func2() {
        out.println("func2()");
    }
}

//////////////// client code ////////////////
Lock[] locks = new Lock[] {
    new BakeryLock(10), new FilterLock("cs")
};

for(Lock lock : locks) {
    lock.lock();
    if(lock <2*> BakeryLock) {
        ((BakeryLock) lock).func1();
    } else if(lock <2*> FilterLock) {
        <3*>
    }
    lock.unlock();
}
```

Question 39

Consider that you have an array of integers of size n named `arr` that you wish to convert into a `PriorityQueue<Integer>` named `pq` by calling `pq.offer(arr[i])` for each `i` in the range 0 through $n - 1$. What is the tightest asymptotic upper bound on this set of operations? Express your answer in Big-O notation in terms of n.

Question 40

Convert the prefix expression to the right into the equivalent fully parenthesized infix expression.

```
* + A - B C / D - E + F G
```

★ANSWER KEY – CONFIDENTIAL★

UIL COMPUTER SCIENCE – 2024-2025 DISTRICT

Questions (+6 points for each correct answer, -2 points for each incorrect answer)

1) E	11) C	21) A	31) C
2) C	12) B	22) D	32) C
3) C	13) D	23) C	33) D
4) D	14) A	24) C	34) B
5) A	15) C	25) A	35) B
6) D	16) B	26) C	36) A
7) B	17) E	27) D	37) C
8) B	18) A	28) J	38) E
9) D	19) D	29) C	*39) $O(n \lg n)$
10) A	20) D	30) B	*40) See Explanation

** See "Explanation" section below for alternate, acceptable answers.*

Note: Correct responses are based on **Java SE Development Kit 22 (JDK 22)** from Sun Microsystems, Inc. All provided code segments are intended to be syntactically correct, unless otherwise stated (e.g., "error" is an answer choice) and any necessary Java SE 22 Standard Packages have been imported. Ignore any typographical errors and assume any undefined variables are defined as used.

Explanations:

1.	E	All of the values are equivalent
2.	C	Simple order of operations problem
3.	C	`printf` is formatting output, “`%3S`”, greater than 3 columns so `3` is ignored, `S` means capitalized.
4.	D	Simple substring problem, substring is inclusive of first value, exclusive of last (if present)
5.	A	Simple `boolean` solving
6.	D	`Math.max(double x, double y)` is called, which returns the larger value, but cast to a `double` since it is the integer value.
7.	B	Simple tracing problem, trace out each value during each iteration of the loop.
8.	B	Simple order of operations problem
9.	D	`else` will not happen if ANY if before it is triggered. In a sequence of `if-else if-else if-else`, only one statement will be activated, no matter how many there are.
10.	A	Array problem, 25 times you move through the array values, but it cycles every 6. $25 \% 6 = 1$, so just do the operation once.
11.	C	`Scanner` class is in the `java.util` package
12.	B	Just trace the loop, you could also estimate and knock out the wrong answers
13.	D	Simple order of precedence
14.	A	`double` = 64 bits = 4 bytes (8 bits each)
15.	C	Add to the end, `ArrayList`s are 0-indexed. Trace it out. Removing 1 will remove the object at index 1, not the `Object` 1 itself.
16.	B	The strings "`instanceof`" and "`int`" first differ on the characters '`s`' and '`t`', respectively. The characters '`s`' and '`t`' have ASCII values of `115` and `116`, respectively. Since we are comparing the first string to the second string, and `s` comes before `t`, and they differ by a single ASCII value, the output of the `compareTo` statement is `-1`. The boolean expression resolves to false, and the ternary operator selects "`Not Greater`" as the return value.
17.	E	The range of the statement is $[21,57)$, and the return type of `Math.random()` is `double`. Both options A and C are contained in this range and are represented as a `double`. Option B, despite being within the valid range, is represented as an `int`, and, therefore, is not a possible value of the expression since it must be a `double`.
18.	A	As previously stated, the range is $[21, 57)$.
19.	D	The graph is directed, which can be noted by the arrow tips, weighted, which can be noted by the weights on the edges, and acyclic, since the graph contains no cycles (flows from node A to node J).
20.	D	All three algorithms are shortest path algorithms that can be used to find the shortest path between a specific pair of nodes. Both Dijkstra’s and Bellman-Ford find the shortest path from a single node to all other nodes, while Floyd-Warshall finds the shortest path between all pairs of nodes (hence, it can be used to find a specific pair, even if it provides more info than what is needed). Dijkstra’s algorithm has a stipulation that the graph must not contain any negative edges. Since the graph contains a single negative edge, Dijkstra’s algorithm is not guaranteed to work (depending on the order that vertices of equivalent potential weight are processed, the query of node A to node B may fail in this case as it may explore vertex B before exploring vertex C). The other two algorithms have a stipulation that the graph must not contain a negative edge cycle reachable from the source. Since the graph is acyclic, this is not an issue.
21.	A	There are two shortest paths from A to J in this graph. Those are $A\ G\ I\ H\ E\ F\ J$, and $A\ C\ B\ D\ E\ F\ J$, both have weight of 18.

22.	D	The following is a copy of the truth table for the expression in question 17:

A	B	C	$\overline{A \oplus \overline{C * B}} + \overline{A \oplus \bar{C} \oplus B}$
F	F	F	F
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

23.	C	You can trace this, but this particular code will set `y` to the value of `x` and `x` to `0`.
24.	C	`super` call to constructor has to be the first line, then `y` must be set to `n`. Only C obeys both of these rules.
25.	A	`get` method will get the `y` variable from class `A`, and the `y` in class `A` attached to `B` will be twice the value given. So, the first `get` call will print `3`, and the second will print `8` because it is called from class `B`.
26.	C	`B` class has no `toString` method, so it will print the memory address, which cannot be determined until runtime.
27.	D	Compile error because the `A` class has no `add` method, and `c` is defined in the eyes of the compiler as an instance of `A`, even though in reality it is an instance of `B`.
28.	J	Note that the symbol after the variable B is known as a "buffer" and simply passes on the value of the input, unaltered. Options A, C, E, and G are equivalent to one another (C is derived by simplifying A, E is derived from simplifying C, and so on). Likewise, options B, D, F, and H are equivalent to one another. The only difference between Options A and B is that Option B incorrectly labels B as $\bar{B}$. Because of this, option J is the correct answer choice.
29.	C	This is an example of DeMorgan's Law which states that $\overline{A + B} = \bar{A} * \bar{B}$.
30.	B	This is an example of the Exclusive NOR Law which states that $\overline{A \oplus B} = A * B + \bar{A} * \bar{B}$
31.	C	Note that the set of nested for loops effectively rotates the square matrix 90 degrees counter-clockwise.
32.	C	See the following recursive table:

Call	Expression	Value
Rec(3,2)	Rec(2,2) + rec(2,1)	1 + 1 = 2
Rec(2,2)	Rec(1,2) + rec(1,1)	0 + 1 = 1
Rec(1,2)	Base Case #1	0
Rec(1,1)	Base Case #2	1
Rec(2,1)	Base Case #2	1

33.	D	You could either create another recursive table for this, or, you could recognize this recursive function as being equivalent to the number appearing on the r^{th} row and c^{th} column of Pascal's Triangle, and quickly determine the value.
34.	B	Reverse engineer the values of r and c by looking at the last return statement in the `rec(r,c)` function definition.
35.	B	As previously stated, this is Pascal's Triangle, where the value appearing on the r^{th} row and c^{th} column of the triangle is equivalent to the sum of the two numbers above it.
36.	A	All options are valid ways to write a method; however, since the `Lock` interface specifies that the visibility of the two methods `lock()` and `unlock()` must be `public`, so too must be the implementations of those methods in the sub-classes.
37.	C	The key word `instanceof` tests to see if a reference variable is an instance of a particular class, or is a subclass of some class or interface. Options A and B are used to show inheritance between classes and interfaces, while Option D is used to call a parent class's constructor.

38.	E	Option A is the equivalent to how `func1()` was called for `BakeryLock` implementations of `Lock`. Option B utilizes the cast method of a `Class` object, which can be statically retrieved from the class literal of that object's type, which is determined at compile time. Option C utilizes an inherited method from the `Object` class which determines the class of an object at runtime, then calls the `getMethod()` method from the `Class` object, and invokes that method on the passed `lock` object. Since option C has the class and method calls determined at runtime, it must be surrounded in a try-catch block. All options are valid ways to call the method `func2()`.
39.	$\mathcal{O}(n \lg n)$ *or* $\mathcal{O}(n \log_2 n)$ *or* $\mathcal{O}(n \log n)$	Each call to `offer()` for a `PriorityQueue` (Java's implementation of a min-heap) takes $\mathcal{O}(\lg n)$ time. This is performed n times, once for each of the n elements of `arr`. This yields $$n \cdot \mathcal{O}(\lg n) = \boxed{\mathcal{O}(n \lg n)}$$
40.	`( ( A + ( B - C ) ) * ( D / ( E - ( F + G ) ) ) )`	
	To convert between prefix and infix, perform the following: 1. Read symbols from right to left 2. If the symbol read is an operand, push it onto a stack. 3. If the symbol read is an operator, pop the two values from the stack. 4. Create a string by concatenating the two operands and the operator, where string = (2nd top-most value <op> 1st top-most value) 5. Repeat until entire string is read and stack only contains a single value. This is your fully parenthesized infix-equation. Note that the order in which variables occur, even for operators that exhibit the commutative and associative properties, is important when translating between the two forms since the rule is generalized for all operators, even those that don't exhibit these properties. Because of this, no other otherwise equivalent forms of this equation will be accepted.	